Comparing implicit characterizations by program transformations

Guillaume Bonfante

Université de Nancy

Workshop on Logic and Computations
8-9th February 2011, Kanazawa

Programs, rules

A program is a set of (confluent) rules:

$$\begin{aligned} \mathtt{not}(\mathbf{tt}) &\to \mathbf{ff} \\ \mathtt{not}(\mathbf{ff}) &\to \mathbf{tt} \end{aligned} \qquad \begin{aligned} \mathtt{or}(\mathbf{tt}, y) &\to \mathbf{tt} \\ \mathtt{or}(x, \mathbf{tt}) &\to \mathbf{tt} \\ \mathtt{or}(\mathbf{ff}, \mathbf{ff}) &\to \mathbf{ff} \end{aligned} \qquad \begin{aligned} \mathbf{0} &= \mathbf{0} &&\to \mathbf{tt} \\ \mathbf{0} &= \mathbf{s}(y) &&\to \mathbf{ff} \\ \mathbf{s}(x) &= \mathbf{0} &&\to \mathbf{ff} \\ \mathbf{s}(x) &= \mathbf{s}(y) &&\to x = y \end{aligned}$$

$$\begin{aligned} \mathtt{in}(x, \mathbf{nil}) &\to \mathbf{ff} \\ \mathtt{in}(x, \mathbf{cons}(a, l)) &\to \mathtt{or}(x = a, \mathtt{in}(x, l)) \end{aligned}$$

Semantics

- $[\![\mathtt{f}]\!] : \mathcal{T}(\mathcal{C})^n \to \mathcal{T}(\mathcal{C})$
- $[\![\mathtt{f}]\!](t_1, \ldots, t_n) = t$ iff $f(t_1, \ldots, t_n) \xrightarrow{!} t$

The complexity of programs

Use:

- program interpretations,
- syntactic properties,
- termination proofs (Product Path Ordering)

Programs, rules

A program is a set of (confluent) rules:

$$\begin{array}{rl} \mathtt{not}(\mathbf{tt}) & \to \mathbf{ff} \\ \mathtt{not}(\mathbf{ff}) & \to \mathbf{tt} \end{array} \qquad \begin{array}{rl} \mathtt{or}(\mathbf{tt}, y) & \to \mathbf{tt} \\ \mathtt{or}(x, \mathbf{tt}) & \to \mathbf{tt} \\ \mathtt{or}(\mathbf{ff}, \mathbf{ff}) & \to \mathbf{ff} \end{array} \qquad \begin{array}{rll} \mathbf{0} & = \mathbf{0} & \to \mathbf{tt} \\ \mathbf{0} & = \mathbf{s}(y) & \to \mathbf{ff} \\ \mathbf{s}(x) & = \mathbf{0} & \to \mathbf{ff} \\ \mathbf{s}(x) & = \mathbf{s}(y) & \to x = y \end{array}$$

$$\begin{array}{rcl} \mathtt{in}(x, \mathbf{nil}) & \to & \mathbf{ff} \\ \mathtt{in}(x, \mathbf{cons}(a, l)) & \to & \mathtt{or}(x = a, \mathtt{in}(x, l)) \end{array}$$

Interpretations of programs

$$\begin{aligned}
(\!|\mathbf{tt}|\!) = (\!|\mathbf{ff}|\!) &= 1 \\
(\!|\mathbf{0}|\!) = (\!|\mathbf{nil}|\!) &= 1 \\
(\!|\mathbf{s}|\!)(x) &= x + 1 \\
(\!|\mathbf{cons}|\!)(x, y) &= x + y + 3 \\
(\!|\mathtt{not}|\!)(x) &= x + 1 \\
(\!|\mathtt{or}|\!)(x, y) &= x + y + 1 \\
(\!|=|\!)(x, y) &= x + y + 1 \\
(\!|\mathtt{in}|\!)(x, y) &= (x + 1)(y + 1)
\end{aligned}$$

$$\begin{aligned}
(\!|\mathbf{tt}|\!) = (\!|\mathbf{ff}|\!) &= 1 \\
(\!|\mathbf{0}|\!) = (\!|\mathbf{nil}|\!) &= 1 \\
(\!|\mathbf{s}|\!)(x) &= x + 1 \\
(\!|\mathbf{cons}|\!)(x, y) &= x + y + 1 \\
(\!|\mathtt{not}|\!)(x) &= x \\
(\!|\mathtt{or}|\!)(x, y) &= \max(x, y) \\
(\!|=|\!)(x, y) &= \max(x, y) \\
(\!|\mathtt{in}|\!)(x, y) &= \max(x, y)
\end{aligned}$$

Programs, rules

A program is a set of (confluent) rules:

$$\begin{array}{rl} \mathtt{not}(\mathbf{tt}) & \to \mathbf{ff} \\ \mathtt{not}(\mathbf{ff}) & \to \mathbf{tt} \end{array} \qquad \begin{array}{rl} \mathtt{or}(\mathbf{tt}, y) & \to \mathbf{tt} \\ \mathtt{or}(x, \mathbf{tt}) & \to \mathbf{tt} \\ \mathtt{or}(\mathbf{ff}, \mathbf{ff}) & \to \mathbf{ff} \end{array} \qquad \begin{array}{rll} \mathbf{0} & = \mathbf{0} & \to \mathbf{tt} \\ \mathbf{0} & = \mathbf{s}(y) & \to \mathbf{ff} \\ \mathbf{s}(x) & = \mathbf{0} & \to \mathbf{ff} \\ \mathbf{s}(x) & = \mathbf{s}(y) & \to x = y \end{array}$$

$$\begin{array}{rcl} \mathtt{in}(x, \mathbf{nil}) & \to & \mathbf{ff} \\ \mathtt{in}(x, \mathbf{cons}(a, l)) & \to & \mathtt{or}(x = a, \mathtt{in}(x, l)) \end{array}$$

Programs, rules

A program is a set of (confluent) rules:

$$\begin{aligned} 2 &> 1 \\ 2 &> 1 \end{aligned} \qquad \begin{aligned} y+2 &> 1 \\ x+2 &> 1 \\ 3 &> 1 \end{aligned} \qquad \begin{aligned} 3 &> 1 \\ y+3 &> 1 \\ x+3 &> 1 \\ x+y+3 &> x+y+1 \end{aligned}$$

$$\begin{aligned} x+2 &> 1 \\ (x+1)(a+l+2) &> x+a+2+(x+1)(l+1) \end{aligned}$$

Programs, rules

A program is a set of (confluent) rules:

$$1 \geq 1 \qquad \max(1, y) \geq 1 \qquad 1 \geq 1$$

$$1 \geq 1 \qquad \max(x, 1) \geq 1 \qquad \max(1, y+1) \geq 1$$

$$1 \geq 1 \qquad \max(x+1, 1) \geq 1$$

$$\max(x+1, y+1) \geq \max(x, y)$$

$$\begin{aligned} \max(x, 1) &\geq 1 \\ \max(x, a + \ell + 1) &\geq \max(x, a, \ell) \end{aligned}$$

Interpretation

Definition

A $(\mathcal{C} \cup \mathcal{F})$-algebra $(\!|-|\!)$ on $\mathbf{N}$ is said to be an additive *strict interpretation* if:

1. for all constructors $\mathbf{c}$, $\mathbf{c}(x_1, \ldots, x_n) = c_{\mathbf{c}} + \sum_{i=1}^{n} x_i$ with $c > 0$,
2. for all f, $(\!|f|\!)$ is a strictly monotonic function, that is if $x_i > x_i'$, then $(\!|f|\!)(x_1, \ldots, x_n) > (\!|f|\!)(x_1, \ldots, x_i', \ldots, x_n)$,
3. for all f, $(\!|f|\!)(x_1, \ldots, x_n) \geq \max(x_1, \ldots, x_n)$,
4. for all rules $\ell \to r$, $(\!|\ell|\!) > (\!|r|\!)$.

Interpretation

Definition

A $(\mathcal{C} \cup \mathcal{F})$-algebra $(\!|-|\!)$ on $\mathbf{N}$ is said to be an additive *quasi-interpretation* if:

1. for all constructors $\mathbf{c}$, $\mathbf{c}(x_1, \ldots, x_n) = c_{\mathbf{c}} + \sum_{i=1}^{n} x_i$ with $c > 0$,
2. for all f, $(\!|f|\!)$ is a weakly monotonic function, that is if $x_i \geq x_i'$, then $(\!|f|\!)(x_1, \ldots, x_n) \geq (\!|f|\!)(x_1, \ldots, x_i', \ldots, x_n)$,
3. for all f, $(\!|f|\!)(x_1, \ldots, x_n) \geq \max(x_1, \ldots, x_n)$,
4. for all rules $\ell \to r$, $(\!|\ell|\!) \geq (\!|r|\!)$.

Strict interpretations vs quasi-interrpretations

$$
\begin{aligned}
\mathtt{max}(n, \mathbf{0}) &\to n \\
\mathtt{max}(\mathbf{0}, m) &\to m \\
\mathtt{max}(\mathbf{s}(n), \mathbf{s}(m)) &\to \mathbf{s}(\mathtt{max}(n, m)) \\
\mathtt{lcs}(\varepsilon, y) &\to \mathbf{0} \\
\mathtt{lcs}(x, \varepsilon) &\to \mathbf{0} \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)) &\to \mathbf{s}(\mathtt{lcs}(x, y)) && \mathbf{i} \in \{\mathbf{a}, \mathbf{b}\} \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)) &\to \mathtt{max}(\mathtt{lcs}(x, \mathbf{j}(y)), \mathtt{lcs}(\mathbf{i}(x), y)) && \mathbf{i} \neq \mathbf{j}
\end{aligned}
$$

$\mathtt{lcs}(\mathbf{ababa}, \mathbf{baaba})$ evaluates to $\mathbf{s}^4(\mathbf{0})$. The length of the longuest common subsequence is 4 (take **baba**).

Strict interpretations vs quasi-interrpretations

It admits the following quasi-interpretation:

- $(\!|\varepsilon|\!) = (\!|\mathbf{0}|\!) = 0$
- $(\!|\mathbf{a}|\!)(X) = (\!|\mathbf{b}|\!)(X) = (\!|\mathbf{s}|\!)(X) = X + 1$
- $(\!|\mathtt{lcs}|\!)(X, Y) = (\!|\mathtt{max}|\!)(X, Y) = \max(X, Y)$

but try to find a polynomial satisfying:

$$\mathtt{lcs}(x+1, y+1) > \mathtt{lcs}(x+1, y) + \mathtt{lcs}(x, y+1)$$

Product Path Ordering with sub-term

Definition

Given a partial order $\preceq_{\mathcal{F}}$ on function symbol, the $\trianglelefteq$-Product Path Ordering $\prec_{ppo}$ is defined by the rules:

$$\frac{t \lhd t'}{t \prec_{ppo} t'} \qquad \frac{s \prec_{ppo} \mathtt{f}(t)}{\mathbf{c}(s) \prec_{ppo} \mathtt{f}(t)} \mathtt{f} \in \mathcal{F}, \mathbf{c} \in \mathcal{C}$$

$$\frac{s \prec_{ppo} \mathtt{f}(t) \qquad \mathtt{g} \prec_{\mathcal{F}} \mathtt{f}}{\mathtt{g}(s) \prec_{ppo} \mathtt{f}(t)} \mathtt{f}, \mathtt{g} \in \mathcal{F}$$

$$\frac{s \prec_{ppo} (t)}{\mathtt{f}(s) \prec_{ppo} \mathtt{f}(t)}$$

Product Path Ordering with embedding

Definition

Given a partial order $\preceq_{\mathcal{F}}$ on function symbol, the $\trianglelefteq$-Product Path Ordering $\prec_{ppo}$ is defined by the rules:

$$\frac{t \blacktriangleleft t'}{t \prec_{ppo} t'} \qquad \frac{s \prec_{ppo} \mathtt{f}(t)}{\mathbf{c}(s) \prec_{ppo} \mathtt{f}(t)} \mathtt{f} \in \mathcal{F}, \mathbf{c} \in \mathcal{C}$$

$$\frac{s \prec_{ppo} \mathtt{f}(t) \qquad \mathtt{g} \prec_{\mathcal{F}} \mathtt{f}}{g(s) \prec_{ppo} \mathtt{f}(t)} \mathtt{f}, \mathtt{g} \in \mathcal{F}$$

$$\frac{s \prec_{ppo} (t)}{\mathtt{f}(s) \prec_{ppo} \mathtt{f}(t)}$$

$\trianglelefteq$-PPO vs $\blacktriangleleft$-PPO

$$1(1(1(\bullet))) \prec^{\blacktriangleleft}_{ppo} 0(1(0(1(0(1(\bullet))))))$$

$$1(1(1(\bullet))) \not\prec^{\trianglelefteq}_{ppo} 0(1(0(1(0(1(\bullet))))))$$

$$4(5(6(\bullet))) \prec^{\trianglelefteq}_{ppo} 1(2(3(4(5(6(\bullet))))))$$

$$4(5(6(\bullet))) \prec^{\blacktriangleleft}_{ppo} 1(2(3(4(5(6(\bullet))))))$$

$\trianglelefteq$-PPO vs $\blacktriangleleft$-PPO

$$1(1(1(\bullet))) \prec^{\blacktriangleleft}_{ppo} 0(1(0(1(0(1(\bullet))))))$$
$$1(1(1(\bullet))) \not\prec^{\trianglelefteq}_{ppo} 0(1(0(1(0(1(\bullet))))))$$
$$4(5(6(\bullet))) \prec^{\trianglelefteq}_{ppo} 1(2(3(4(5(6(\bullet))))))$$
$$4(5(6(\bullet))) \prec^{\blacktriangleleft}_{ppo} 1(2(3(4(5(6(\bullet))))))$$

$$\begin{aligned}
f(0(x)) &\to f(x) \\
f(1(0(x))) &\to f(1(x)) \\
f(1(1(x))) &\to \mathbf{tt} \\
f(\bullet) &\to \mathbf{ff} \\
f(1(\bullet)) &\to \mathbf{ff}
\end{aligned}$$

Cons-free programs

Definition

For all rule $f(p_1, \cdots, p_n) \to r$, for all sub-term $t \trianglelefteq r$,

- either t is a constructor term and
  - either $t \trianglelefteq f(p_1, \cdots, p_n)$,
  - or t is a ground term (without variable).
- or the root of t is not a constructor symbol.

Programs, rules

A program is a set of (confluent) rules:

$$\begin{aligned} \mathtt{not}(\mathbf{tt}) &\to \mathbf{ff} \\ \mathtt{not}(\mathbf{ff}) &\to \mathbf{tt} \end{aligned} \qquad \begin{aligned} \mathtt{or}(\mathbf{tt}, y) &\to \mathbf{tt} \\ \mathtt{or}(x, \mathbf{tt}) &\to \mathbf{tt} \\ \mathtt{or}(\mathbf{ff}, \mathbf{ff}) &\to \mathbf{ff} \end{aligned} \qquad \begin{array}{rll} \mathbf{0} & = \mathbf{0} & \to \mathbf{tt} \\ \mathbf{0} & = \mathbf{s}(y) & \to \mathbf{ff} \\ \mathbf{s}(x) & = \mathbf{0} & \to \mathbf{ff} \\ \mathbf{s}(x) & = \mathbf{s}(y) & \to x = y \end{array}$$

$$\begin{array}{rcl} \mathtt{in}(x, \mathbf{nil}) & \to & \mathbf{ff} \\ \mathtt{in}(x, \mathbf{cons}(a, l)) & \to & \mathtt{or}(x = a, \mathtt{in}(x, l)) \end{array}$$

Four programming languages

- CF-programs are constructor-free programs;
- SI-programs are programs which admit a strict additive interpretation;
- $\trianglelefteq$-MI-programs are programs which admit a quasi-interpretation and a proof of termination by $\trianglelefteq$-PPO.
- $\underline{\blacktriangleleft}$-MI-programs are programs which admit a quasi-interpretation and a proof of termination by $\underline{\blacktriangleleft}$-PPO.

Complexity characterizations

Theorem (following N. Jones)

Predicates computed by CF-programs are exactly $\mathrm{P}\textsc{time}$ *predicates.*

Complexity characterizations

Theorem (following N. Jones)

Predicates computed by CF-programs are exactly PTIME *predicates.*

Theorem (B., Cichon, Marion, Touzet)

Predicates computed by SI-programs are exactly PTIME *predicates.*

Complexity characterizations

Theorem (following N. Jones)

Predicates computed by CF-programs are exactly PTIME *predicates.*

Theorem (B., Cichon, Marion, Touzet)

Predicates computed by SI-programs are exactly PTIME *predicates.*

Theorem (B., Marion, Moyen)

Predicates computed by $\trianglelefteq$*-MI-programs are exactly* PTIME *predicates.*

Complexity characterizations

Theorem (following N. Jones)

Predicates computed by CF-programs are exactly $\textsc{Ptime}$ *predicates.*

Theorem (B., Cichon, Marion, Touzet)

Predicates computed by SI-programs are exactly $\textsc{Ptime}$ *predicates.*

Theorem (B., Marion, Moyen)

Predicates computed by $\trianglelefteq$*-MI-programs are exactly* $\textsc{Ptime}$ *predicates.*

Theorem

Predicates computed by $\blacktriangleleft$*-MI-programs are exactly* $\textsc{Ptime}$ *predicates.*

Complexity characterizations

Theorem (following N. Jones)

Predicates computed by CF-programs are exactly $\textsc{Ptime}$ *predicates.*

Theorem (B., Cichon, Marion, Touzet)

Predicates computed by SI-programs are exactly $\textsc{Ptime}$ *predicates.*

Theorem (B., Marion, Moyen)

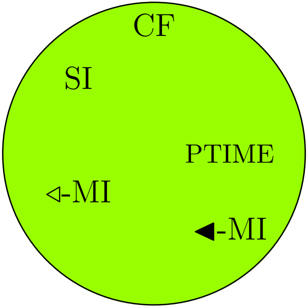
Predicates computed by $\trianglelefteq$*-MI-programs are exactly* $\textsc{Ptime}$ *predicates.*

Theorem

Predicates computed by $\underline{\blacktriangleleft}$*-MI-programs are exactly* $\textsc{Ptime}$ *predicates.*

In other words, CF $\simeq$ *SI* $\simeq$ MI$\trianglelefteq$ $\simeq$ MI$\underline{\blacktriangleleft}$.

Summary



Transformations of programming languages

Apply:

- Non-confluence
- Course of value

to the four programming languages CF, SI, $\mathrm{MI}_{\trianglelefteq}$, $\mathrm{MI}_{\blacktriangleleft}$.

Non confluence

Definition

Let $\mathcal{L}$ be a set of TRS, $\mathcal{L}$.N is the set of TRS $(R_1 \cup R_2)$ such that $R_1 \in \mathcal{L}$ and $R_2 \in \mathcal{L}$.

Non confluence

Definition

Let $\mathcal{L}$ be a set of TRS, $\mathcal{L}$.N is the set of TRS $(R_1 \cup R_2)$ such that $R_1 \in \mathcal{L}$ and $R_2 \in \mathcal{L}$.

Definition

- $[\![\mathtt{f}]\!] : \mathcal{T}(\mathcal{C})^n \to \mathcal{T}(\mathcal{C})$
- $[\![\mathtt{f}]\!](t_1, \ldots, t_n) = t$ iff $f(t_1, \ldots, t_n) \xrightarrow{!} t$

Non confluence

Definition

Let $\mathcal{L}$ be a set of TRS, $\mathcal{L}$.N is the set of TRS $(R_1 \cup R_2)$ such that $R_1 \in \mathcal{L}$ and $R_2 \in \mathcal{L}$.

Definition

- $[\![\mathtt{f}]\!] : \mathcal{T}(\mathcal{C})^n \to \mathcal{T}(\mathcal{C})$
- $[\![\mathtt{f}]\!](t_1, \ldots, t_n) = t$ iff $f(t_1, \ldots, t_n) \xrightarrow{!} t$

Definition

- $[\![\mathtt{f}]\!] : \mathcal{T}(\mathcal{C})^n \to \mathcal{T}(\mathcal{C})$
- $[\![\mathtt{f}]\!](t_1, \ldots, t_n) = \max\{t \in \mathcal{T}(\mathcal{C}) \mid f(t_1, \ldots, t_n) \xrightarrow{!} t\}$

Complexity characterizations

Theorem

Predicates computed by CF.N programs are exactly PTIME *predicates.*

Theorem (B., Cichon, Marion, Touzet)

Predicates computed by SI.N-programs are exactly NPTIME *predicates.*

Theorem

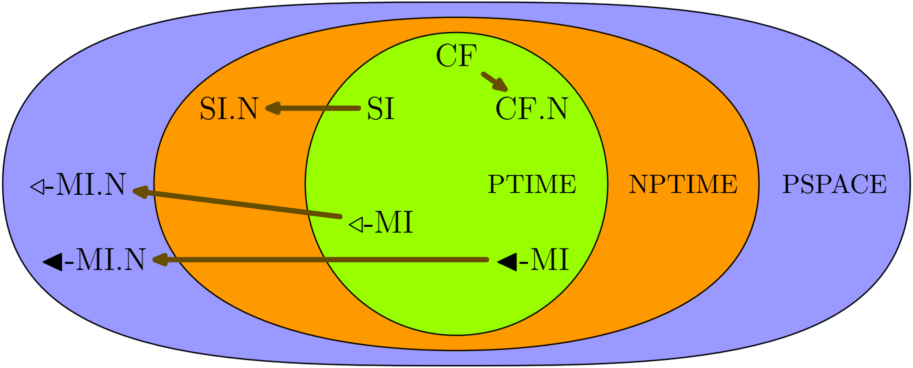
Predicates computed by $\trianglelefteq$*-MI.N-programs are exactly* PSPACE *predicates.*

Theorem

Predicates computed by $\underline{\blacktriangleleft}$*-MI.N-programs are exactly* PSPACE *predicates.*

In other words, Cons-free $\not\simeq$ *SI* $\not\simeq$ MI$\trianglelefteq$ $\simeq$ MI$\underline{\blacktriangleleft}$.

Summary



Course-of-value : Peter, Rose

$$\begin{aligned} f(0,x) &= g(x) \\ f(n+1,x) &= h(n,x,f(j(n),x)) \text{ avec } j(n) \leq n \end{aligned}$$

Theorem (Peter)

If g, h et j are primitive recursive, so is f.

However, the proof involves the encoding of sequences, thus, we are "above" exponential time.

Cov-extension of programs

One suppose that there is an extraterritorial relation R defining $\mathcal{L} = \{p \mid f(u_1, \cdots, u_n) \to s \Rightarrow (f(u_1, \cdots, u_n), s) \in R\}$.

Cov-extension of programs

One suppose that there is an extraterritorial relation R defining $\mathcal{L} = \{p \mid f(u_1, \cdots, u_n) \to s \Rightarrow (f(u_1, \cdots, u_n), s) \in R\}$.

Definition

$\mathcal{L}$.cov is the set of programs such that:
$f(u_1, \cdots, u_n) \to C[f_1(u_1^1, \ldots, u_m^1), \ldots, f_k(u_1^k, \ldots, u_{m_k}^k)]$

- $f_i \simeq f$,
- C does not contain symbols equivalent to f,
- the u_i^js do not contain symbols equivalent to f,

$(f(u_1, \cdots, u_n), f_i([\![u_i^1]\!], \ldots, [\![u_m^i]\!])) \in R$.

Our notion covers recurrence with parameter substitution

$$
\begin{array}{rcl}
f(0,x) & = & g(x) \\
f(n+1,x) & = & h(n,x,f(n,j_1(x)),f(n,j_2(x)))
\end{array}
$$

Theorem (Peter)

If g, h, j_1 and j_2 are primitive recursive, so is f.

Our notion covers recurrence with parameter substitution

$$
\begin{aligned}
f(0,x) &= g(x) \\
f(n+1,x) &= h(n,x,f(n,j_1(x)),f(n,j_2(x)))
\end{aligned}
$$

Theorem (Peter)

If g, h, j_1 and j_2 are primitive recursive, so is f.

Theorem (Leivant/Bellantoni and Cook)

Ramified programs characterize PTIME.

Our notion covers recurrence with parameter substitution

$$\begin{aligned} f(0,x) &= g(x) \\ f(n+1,x) &= h(n,x,f(n,j_1(x)),f(n,j_2(x))) \end{aligned}$$

Theorem (Peter)

If g, h, j_1 and j_2 are primitive recursive, so is f.

Theorem (Leivant/Bellantoni and Cook)

Ramified programs characterize PTIME.

Theorem (Leivant and Marion)

Ramified programs with parameter substitution characterize PSPACE.

Complexity characterization

Theorem

Predicates computed by CF.cov programs are exactly $\textsc{Ptime}$ *predicates.*

Theorem

Predicates computed by SI.cov-programs are exactly $\textsc{Ptime}$ *predicates.*

Theorem

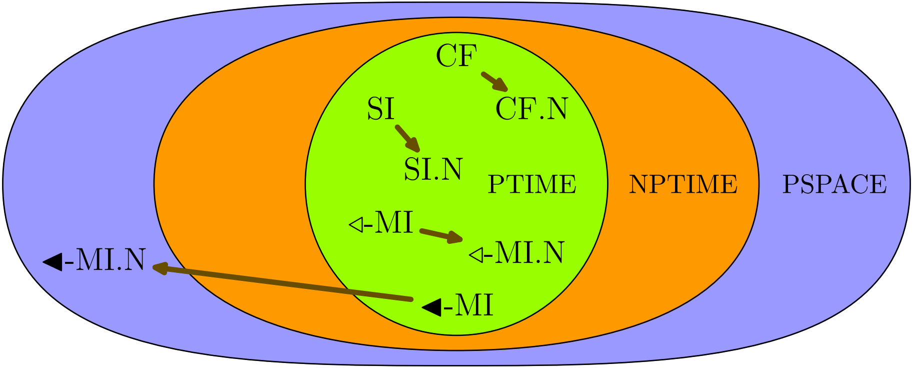
Predicates computed by $\trianglelefteq$*-MI.N-programs are exactly* $\textsc{Ptime}$ *predicates.*

Theorem

Predicates computed by $\underline{\blacktriangleleft}$*-MI.N-programs are exactly* $\textsc{Pspace}$ *predicates.*

In other words, Cons-free $\simeq$ *SI* $\simeq$ MI$\trianglelefteq$ $\not\simeq$ MI$\underline{\blacktriangleleft}$.

Summary



To go on

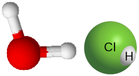
- Example on the logical side
- POP and WIDG (Moser and Hirokawa) compute $\textsc{Ptime}$, but which $\textsc{Ptime}$?

Thanks to

the JAIST, Nao Hirokawa,

Laurent Bringel and the Lycée Henri Poincaré

Thanks to

the JAIST, Nao Hirokawa, 

Laurent Bringel and the Lycée Henri Poincaré